RESP: A Computer Aided OWL REasoner Selection Process

Wei Tai, John Keeney, Declan O'Sullivan
Knowledge and Data Engineering Group, School of Computer Science and Statistics,
Trinity College Dublin, Dublin 2, Ireland
{TaiW, John.Keeney, Declan.OSullivan}@cs.tcd.ie

Abstract— **Existing approaches for selecting the most appropriate reasoner for different semantic applications mainly relies on discussions between application developers and reasoner experts. However this approach will become inadequate with the increasing adoption of Semantic Web technologies in applications from different domains and the rapid development of OWL reasoning technologies. This work proposes RESP, a computer aided reasoner selection process designed to perform reasoner selection for different applications and so reduce the effort and communication overhead required to select the most appropriate reasoner. Preliminary evaluation results show that RESP successfully helps application developers to select the most appropriate reasoner, or at least narrow down the number of candidate reasoners to consider. Contributions of this work are two folds: (1) the design of a (relatively simple but useful) computer aided OWL reasoner selection process, and (2) the identification and discussion of a set of example application characteristics that can affect the OWL reasoner selection.**

Keywords- Reasoner Selection, Application Characteristics, Reasoner Characteristics, Semantic Application Development

I. Introduction

The formal definition of the Web Ontology Language (OWL) enables automated reasoning or entailment to be performed over the contents of OWL ontologies revealing hidden knowledge. Owl has been adopted in ever more applications from various domains, such as bioinformatics [5], publish/subscribe systems [3, 7, 21], configuration management [15], sensor networking [2, 4, 14, 16] and so on. This, conversely, has stimulated extensive studies on OWL reasoning technologies, with an ever growing number of new OWL reasoners and reasoning technologies emerging.

However specific reasoners are designed to solve different problems, such as: complete OWL-DL classification, efficient conjunctive query answering, reasoning over large datasets and so on, and reasoners may need to trade off one reasoner characteristic (RC) against another, depending on their specialisation. For example a reasoner designed for efficient and scalable ABox query answering may not compute complete OWL-DL classification therefore is not suitable for applications requiring complete OWL-DL classification. Similarly an in-memory reasoner capable of effectively and completely classifying ontologies may not natively support file-backed storage therefore is not suitable for applications with large datasets when used on its own. It is clear from the above examples that interplays between RCs and the application characteristics (AC) determine the suitability of a reasoner for a specific application.

The existing reasoner selection approach relies largely on consultation between application developers and reasoner experts, and until recently this approach was straightforward and sufficient because of the relatively small number of OWL reasoners and RCs. However, the ever widespread adoption of semantic web technologies for applications in different domains and the rapid development and emergence of new OWL reasoning technologies cause this approach to become increasingly inadequate in the future. Firstly, as semantic applications grow more complicated and move beyond initial prototyping stages, these applications will be developed and extended by dedicated application developers with little or no knowledge of the intricacies of ontology reasoning. Furthermore reasoner experts may not always precisely understand some ACs expressed in domain specific languages. All these could result in a considerable amount of effort to be devoted before an agreement is reached, or even a wrong reasoner to be selected. Secondly the existing approach requires that a reasoner expert is accessible to application developers, which will not always be the case. These inadequacies motivate an automated approach to help application developers to limit consultation requirements or even to independently select a suitable reasoner for their semantic applications.

This paper describes RESP, a computer aided OWL REasoner Selection Process, to select an appropriate reasoner for applications, based on the particular ACs of the application. RESP matchmakes between ACs and RCs. Application developers input the set of ACs for their application, and by matching them to the RCs of registered candidate reasoners, according to a set of predefined AC/RC connections, RESP evaluates and ranks the suitability of candidate reasoners. A prototype implementation of RESP, termed TARS, is described for demonstration and evaluation purposes.

A usability experiment is carried out to evaluate RESP using TARS. RESP helps participants with little knowledge of ontology reasoning to automatically select an appropriate reasoner or reduce the number of candidate reasoners to consider. Analysis of post experiment questionnaires indicates that with some further development it is likely that most participants could have independently identified the appropriate reasoner. (Note: this work is based on OWL 1, and extensions of this work for OWL2 are ongoing and will be presented in future work.)

This paper is organized as follows. Section 2 presents work related to this research. Section 3 details RESP and a set of example ACs. TARS is presented in section 4, followed by evaluation and discussion in section 5. This work concludes in section 6 with a discussion of ongoing and future work.

II. Related Work

To the best of our knowledge there is no similar research on this topic. Therefore related work of this research is comprised of a discussion of sources where ACs/RCs are identified, including a survey of ontology-based applications from four candidate domains and a categorization of OWL reasoners.

A. *Survey of Ontology-based Applications*

This section briefly discusses how OWL and its reasoning technologies are applied in applications from four areas: publish/subscribe (pub/sub) systems, sensor network systems, configuration management systems, and bioinformatics/ medical systems.

1) Semantic Publish/Subscribe Systems: In pub/sub systems subscribers register subscriptions in a broker (or networked brokers) and publishers present publications to the broker. A conventional pub/sub broker syntactically matches the content of publications against registered subscriptions and successfully matched publications are propagated to the corresponding matched subscribers. Semantic pub/sub systems extend this approach by matching based on the semantics of message contents, informed by an associated ontology and facilitated by an ontology reasoner in the broker.

Research conducted in [7] presents a document retrieval service based on a (centralized) semantic pub/sub system, implemented using the Racer system [6]. This research points out the importance of closed-world queries in information retrieving systems and uses set operations to simulate a local closed world in an open world environment. In addition an incremental ABox query answering mechanism is adopted to avoid continuous querying evaluation. Later work in [3] points out the scalability problem of DL tableau reasoners when applied in semantic pub/sub systems [7, 18], i.e. re-checking consistency of the knowledge base (KB) from scratch and re-evaluating subscriptions for each update. It proposes to use incremental consistency checking and incremental query answering as a solution. Work in [21] presents a series of use-cases for semantic publish/subscribe systems, demonstrating that even for a single middleware system, different reasoners may be appropriate for different deployments.

2) Semantic Sensor Network Systems: Semantic Web technologies are widely applied in sensor network systems. A typical usage is annotating sensor readings (or sensors descriptions) using semantically rich tags enabling more intelligent data processing [2, 24, 36] and better interoperability [14, 16, 35]. Another usage could be the use of ontology and ontology reasoning technologies to perform complex management tasks, e.g. sensor tasks assignments [4] or fault correlation [41].

Work in [2] describes a coastal ecosystem monitoring application using wireless sensor networks for data collection and delivery, and an OWL reasoner for data validation and inference. Other than standard OWL reasoning services several other features are also required in this system, including the use of rules to model domain specific knowledge and numeric comparison/computation. This research also extracts two reasoning requirements for sensor network systems, including the requirement for distributed reasoning and the provision of a user-friendly graphical interface for domain knowledge authoring. Research in [4] presents a semantic sensor task assignment approach for the intelligence, surveillance and reconnaissance domain. This work observes the high degree of variability in such environment and therefore requires any solutions to sensor task assignment in this domain should be of high agility against changes. In addition they also point out that explanation of assignments is required. The Semantic Sensor Web (SSW) [16] suggests enriching sensor observations with semantic metadata to enhance interoperability, with some of the semantic processing taking place on embedded devices with limited resources. Web-aware approaches are necessary for both management and data processing and rule-based reasoning should be used to derive new knowledge based on application-specific semantics. Work in [20] surveys over a set of 12 sensor ontologies and points out, conjunctive queries, rules and OWL reasoning were key technologies to provide semantics support at different layers in semantic sensor networks.

3) Configuration Management Systems: Configuration management is another area that adopts ontology and ontology reasoning techniques. Work in [15] proposes using OWL inconsistency checking to deduce the validity of software configuration. The DL $\mathcal{ALCO}$ subset is found to be sufficient for their modelling, however a full-fledged OWL-DL reasoner is still used in order to completely deduce inconsistencies. In addition justification of validity is also required. Work in [52] also shows the need to support domain-specific rules in this application area.

4) Bioinformatics and Medical Systems: Ontology-based approaches are widely applied in bioinformatics for knowledge access, modelling, and reasoning. Two well-known applications are the Gene project[1], which provide a controlled vocabulary of terms (concepts) for describing genes and gene product attributes [5], and the SNOMED ontology[2], to provide a scientifically validated set of terms for practitioners to share health care knowledge.

Research in [11] identifies nine requirements that OWL-based bio-ontologies may have on OWL reasoning. They are: supporting the ontology development process; classification; model checking; finding gaps in an ontology and discovering new relations; comparison of ontologies; reasoning with mereological parthood and other (part-whole) relations; using a hierarchy of relations; reasoning across linked ontologies; and complex queries. Some of these, e.g. classification, can be

[1] http://www.geneontology.org/

[2] http://www-calit2.nbirn.net/research/ontology/snomed.shtm

solved by existing OWL reasoners, whereas some others, e.g. finding gaps and new relations, were quite specific to life science and were not yet feasible for existing OWL reasoners.

B. *OWL Reasoner Categorization*

We selected a subset of available OWL reasoners and categorized them into four types according to their reasoning algorithms: forward-chaining *rule-entailment reasoners*, *resolution-based reasoners, DL tableaux reasoners*, and *hybrid reasoners*. The first two types together are referred to as rule-based reasoners in this paper as they are (horn) rule-based. Resolution-based reasoners are further divided into two subtypes, semantic reduction reasoners and syntactic reduction reasoners, according to the way OWL is reduced.

In the following subsections example reasoners are listed for each reasoner type. Although some of the selected reasoners are not currently in active development, they are included as they are well-documented and are typical reasoners of the type, and their source code is publically accessible.

1) Forward-Chaining Rule-Entailment Reasoners: rule-entailment reasoners reason over OWL ontologies using a set of entailment rules. Rules are evaluated in forward chaining paradigm using algorithms such as RETE [22] or RDBMS. Reasoners of this type usually have the intrinsic capability to process rules and usually have support for closed-world assumption (CWA). Furthermore some reasoners of this type are designed to enable (relatively) scalable data stores to be built and reasoned, e.g. OWLIM[3] is able to process ontologies as large as billions of triples. However rule-entailment reasoners cannot provide complete OWL DL reasoning. Other rule entailment reasoners include Jena[4] and BaseVISor[5].

2) Backward-Chaining Resolution-Based Reasoners: Resolution-based reasoners reduce OWL ontology to First Order Logic (FOL) programs or FOL fragments (e.g. horn clauses), and delegate OWL reasoning to a resolution engine such as FOL theorem prover or (disjunctive) Datalog engine.

The reduction from OWL ontology to FOL programs can follow either a semantic or a syntactic approach. The former approach reduces OWL to FOL by following their semantic connections. For example the OWL axiom *subclassOf*(C,D) is reduced to FOL clause as $\neg C^T(x) \vee D^T(x)$, where C^T and D^T are the FOL predicates for C and D. Reasoners adopting this approach include KAON2 [12] and Hoolet [17]. The latter approach syntactically translates OWL constructs into FOL, e.g. the above axiom is reduced to $subclassOf^T$(C, D) where $subclassOf^T$ is a syntactic FOL-translation of *subclassOf*. Entailment rules are still required to ensure reasoning. Reasoners of this type include Orel [51] and F-OWL [19].

3) DL tableaux reasoners: DL tableaux reasoners convert OWL ontology into Description Logic (DL) formulae and then perform reasoning by checking KB satisfiability [1]. Some DL Tableaux reasoners, such as Pellet[6], FaCT++[7] and RacerPro[8], are able to perform complete OWL DL reasoning. Some DL tableaux reasoners such as Pellet are also extended to support rules and CWA.

4) Hybrid Reasoners: some reasoners combine more than one reasoning algorithms. DLEJena [28] combines the DL tableau reasoning algorithm with rule-entailment algorithm by using the former to perform efficient and complete TBox reasoning while using the later to perform scalable ABox reasoning. Hermit [50] tries to limit the or-branching by using a bybrid of resolution and tableau.

There are some other reasoners using algorithms other than any of the above mentioned are not included in this categorization. Some examples include CEL [37], a classifier using a special subsumption algorithm to compute classify the EL++ subset of OWL [31], QuOnto [33] that reformulates users' queries by incorporating TBox information, and SPIN[9] that use SPARQL to model and evaluate OWL 2 RL rules.

Other reasoner categorizations also exist. A categorization can be found in [19], where three types are derived: reasoners using specialized DL engines, reasoners using full FOL theorem provers, and reasoners using a reasoner designed for FOL subsets. This categorization does not distinguish rule-entailment reasoners from resolution-based reasoners (it is included in the third type due to the use of horn rules). However reasoners may vary in some RCs, which may affect their selection in the context of this research. For example semantic-reduction reasoners, e.g. KAON2, can perform query-time reasoning enabling faster response time for changes in KB while rule-entailment reasoners, e.g. Jena, require pre-reasoning before a query is answered.

III. THE RESP REASONER SELECTION PROCESS

This section describes RESP, and identified a set of example ACs. In-depth discussion is provided for each AC.

A. *Overview of RESP*

Before RESP starts functioning some prerequisites are required (Figure 1). A set of candidate ACs need to be identified and reasoners are registered as sets of RCs. Connections between ACs and RCs are analyzed and modelled in RESP allowing the selection process to function. Reasoner selection in RESP is performed in three steps (Figure 1). (1): ACs for the target application are identified and input into RESP. (2): Matching is performed between ACs/RCs according to the predefined connections. (3): Reasoners are ranked according to the number of matched ACs (*satisfaction rate*). A reasoner that matches all input ACs is a suitable reasoner for this application. If no suitable reasoner is found users can revise the previously input ACs and re-run RESP.

3 http://www.ontotext.com/owlim/
4 http://jena.sourceforge.net/
5 http://vistology.com/basevisor/basevisor.html

6 http://clarkparsia.com/pellet/
7 http://code.google.com/p/factplusplus/
8 http://www.racer-systems.com/
9 http://composing-the-semantic-web.blogspot.com/2009/01/owl-2-rl-in-sparql-using-spin.html

RESP is designed to be an abstract process without specifying any technical details for ACs, RCs and the matching algorithm. This enables a domain-specific RESP to be constructed for each domain of applications. Therefore more accurate domain specific vocabularies can be used to describe ACs allowing more targeted use by application developers. In this research we identify and discuss a set of example candidate ACs based on the semantic application survey conducted in section II to evaluate and demonstrate RESP.

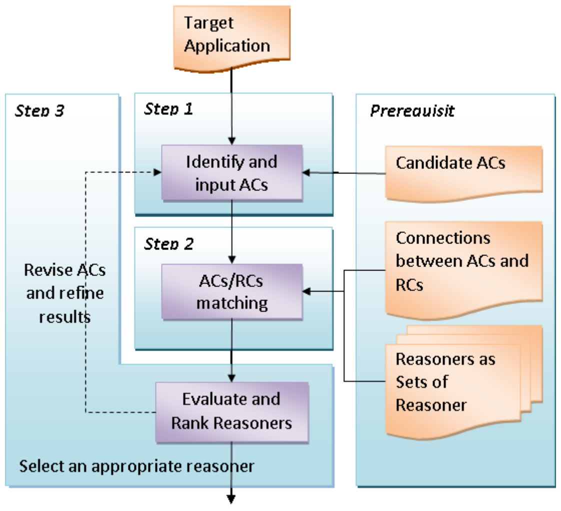
Target Application
Step 3
Step 1
Prerequisit
Identify and input ACs
Candidate ACs
Step 2
Revise ACs and refine results
ACs/RCs matching
Connections between ACs and RCs
Reasoners as Sets of Reasoner
Evaluate and Rank Reasoners
Select an appropriate reasoner

Figure 1. An overview of the RESP reasoner selection process.

B. *Example Candidate ACs*

In this section we discuss the how reasoner selection can be affected by ACs from a selected set of eleven aspects, as listed below. Four things need to be clarified. First this set of ACs is still at its early stage and therefore it is not trying to be complete and definitive. Moreover ACs may vary from domains with different priorities. In this research only some generally applicable ACs are investigated. Thirdly we aim to make only some suggestive connections between ACs and how reasoners can satisfy them. The relative priorities and weightings of different ACs are dependent on the domain and the priorities of the application developer. Lastly connections between ACs and RCs are studied only in analytical rather than empirical means. Therefore performance-related and non-functional issues are not discussed in this research. However we do not deny their importance in selecting an appropriate reasoner for semantic applications and taking the performance of reasoners as an AC is part of the future work.

1) *Reasoning Over Frequently Changing KB*

Many applications, such as semantic pub/sub systems [3, 7, 8] and semantic sensor network systems [4, 24], need to reason over frequently changing KBs rather than static KBs. Therefore the ability to handle changing KBs is important for the underlying reasoners. Most state of the art DL reasoners are designed to reason over static KBs. Some dedicated approaches are proposed allowing the incremental handling of updates of different types. Incremental consistency checking algorithms [8] are proposed to incrementally update tableau completion graphs for ABox updates in DL $\mathcal{SHIQ}$ and $\mathcal{SHOQ}$, subsets of OWL DL without nominals (for $\mathcal{SHIQ}$) and inverse properties (for $\mathcal{SHOQ}$). Later work in [3] addresses the continuous query answering on KBs in DL $\mathcal{SHI}$. Incremental classifications algorithms are also proposed to handle TBox updates [13, 23]. Despite the extensive research of incremental reasoning for DL tableaux algorithms, its application is quite limited. Pellet supports incremental classification but only though OWL API[10] (or Jena) via a specialized reasoner interface. This interface only supports queries about classes and not instance related queries. Pellet's incremental consistency checking is not stable or well tested. Some other reasoners, FaCT++ and CEL claim undocumented support for incremental classification.

Maintaining materialization incrementally for frequently changing KB is a vital issue for rule-based reasoners when reasoning over changing KBs. Work in [25] investigates an approach to compute a complete and correct materialization for rule-based reasoners when streaming data arrive and change KBs. Some reasoner independent technologies are also studied. For example a cascading reasoner structure is proposed in [24] where complex reasoners such as DL tableaux reasoners are layered on top to handle less frequently changing data and relatively simple but efficient rule-based reasoners are layered beneath for maintaining materialization or instantiating certain goal predicates.

2) *Concept- vs. Individual-Centric Reasoning Tasks*

The required reasoning tasks may vary from applications: some bioinformatics/medical applications such as the Gene ontology and SNOMED ontology require TBox classifications while some applications such as semantic sensor network systems need to reason over KBs mainly consisting of (large numbers of) individuals. This feature may largely impact on the selection of an appropriate reasoner as the distinct design goals of different reasoners. Concept centric applications often require a complete concept hierarchy to be constructed and therefore the selected reasoner need to be able to completely classify the ontology. Many DL Tableaux reasoners are designed to provide complete classification reasoning services for OWL DL semantics, such as Pellet, Fact++, and RacerPro. Some other reasoners such as CEL can compute a complete classification for some subsets of OWL.

However DL Tableaux reasoners usually perform poorly when handling large volumes of individual data [26], which is, however, an important feature for individual centric applications. Therefore for individual centric applications rule-based reasoners show better suitability due to their superiority in scale and speed when handling large ABoxes [12, 28, 29].

3) *Required Expressivity*

The level of expressivity required by a semantic application can affect the reasoner to be selected. For example reasoning an $\mathcal{SHOIN(D)}$ ontology as expressive as the wine ontology[11] will usually require a full-fledged reasoner that covers the entire OWL DL semantics, e.g. Pellet or FaCT++, whereas an $\mathcal{ALN}$ ontology can be classified using a relatively simple

10 http://owlapi.sourceforge.net/

11 http://www.w3.org/TR/2003/PR-owl-guide-20031209/wine#

structural subsumption algorithm [1]. This characteristic can also be embodied when the ontology expressivity fall into some specifically designed OWL sublanguages: some rule-entailment reasoners such as OWLIM and BaseVISor use the $p\mathcal{D}*$ semantics family [30]. CEL classifies on DL $\mathcal{EL}$++ (into which many bioinformatics ontologies fall). Owlgres [32] and QuOnto supports efficient query answering over the OWL DL-lite family [34]. Furthermore the expressivity of the ontology is also an important characteristic for composable reasoners to compose its reasoning capability to reduce resource usage [27].

4) *Query-Related Issues*

The means in which applications query ontologies vary. Some applications/ontologies pose complex queries in query languages [14, 20, 35, 36, 40] while some others only need simple queries [21]. This characteristic limits the selection of some reasoners. Reasoners such as Pellet, KAON2, RacerPro, Jena (with ARQ), OWLIM and so on support conjunctive queries to be evaluated. However some reasoners, e.g. CEL and FaCT++, only allow ontologies to be queried atomically using either pre-defined directives at the command line or from its API. They are not suitable for cases where complex queries are required. In addition the syntax and functionality are also distinct for different query languages. SPARQL, one of the most widely used query languages, uses a RDF-based triple syntax. It is (partly) supported by many state of the art reasoners such as Pellet, KAON2, Jena (with ARQ) and RacerPro and so on. Some reasoners also use their own query languages. The nRQL is an axiom-based ABox conjunctive query language specifically designed for RacerPro. OWLIM supports SeRQL, a RDF query language. KAON2 enables queries to be formulated using F-logic. Bossam uses Buchingae. Thea [38] supports queries to be authored using Prolog rules. Some query languages, such as C-SPARQL [39], are implemented however not yet incorporated into state of the art reasoners so they are not discussed here.

5) *Rules*

Other than OWL semantics the ability to model application specific semantics is also an important feature for many applications [2, 16, 20, 41]. Therefore the ability to process rules is important for these applications to function properly. Although varying in the syntax and expressivity of rule languages rule-based reasoners have intrinsic support to model and process rules. Many tableaux reasoners are also extended to support rules, e.g. Pellet and RacerPro partly supports SWRL. However there is no evidence that FaCT++ can process rules and therefore it is not appropriate for such applications. The expressivity and syntax differences among rules supported by different reasoners can also affect the selection of reasoners. For example in some cases the requirement of negation as failure in rules could invalidate reasoners supporting only SWRL rules. Details of differences are not discussed here.

6) *Concrete Domains*

It is widely accepted many real-world applications such as sensor network systems need to handle concrete objects such as strings, numeric numbers, and time. Therefore the ability to reason over concrete domains will play an important role in the selection of an appropriate reasoner for these applications. OWL-DL supports datatypes, a restricted concrete domain. Datatype reasoning has been studied for reasoning algorithms [42, 43, 44, 45] and most state of the art OWL reasoners support reasoning over (part of) the XSD datatypes defined in OWL. However only several of them, including Pellet and Jena, support reasoning over user-defined datatypes.

Another form of concrete domain is algebraic comparison and computations. The extension of DL tableaux algorithm and resolution to support them has been extensively studied [42, 43, 44, 45]. However as OWL does not support modelling predicates over concrete domains (i.e. how concrete objects are related, e.g. comparison, math operations) these features cannot be directly used with OWL. In fact many rule languages have (limited) support through defining built-ins. SWRL provides predefined built-ins to handle algebraic comparison and computations. Therefore a SWRL enabled reasoner is (to some extent) able to process algebraic comparison and computation. Some rule-entailment reasoners, e.g. Jena and Bossam, support user-defined built-ins and therefore allow arbitrary algebraic computations to be modelled.

7) *Closed-World Features*

Some applications, e.g. database-based systems, usually believe the knowledge they possess is a complete modelling of the domain and some reasoning can be performed based on the CWA, e.g. negated queries, checking integrity constraints and so on, where missing information is considered as false. However OWL is based on the open world assumption (OWA), where missing information is treated as unknown, therefore, (may) leading to inferences rather than violation. Much research is devoted to extending OWL with closed-world features [46, 47] and indeed many state of the art reasoners choose to include CWA mainly by two means. First some closed-world features such as negation as failure are introduced through rules or query sub-systems, such as SPARQL, nRQL, SeRQL or Jena rules and so on. Therefore closed-world queries or integrity constraints can be achieved using queries or rules while OWL retains OWA. Second (part of) OWL is modified or extended to use alternative closed-world semantics so it can be used as a language for integrity checking, e.g. Pellet ICV. There is no evidence that reasoners such as FaCT++ and CEL have internal support for closed-world features. However they can be connected to Protégé where SPARQL queries are posed.

8) *Database*

The requirement to store and reason over large datasets is often required by data-centric applications such as sensor networks and bioinformatics applications (e.g. Gene ontology) where in-memory ontology reasoning could exhaust available memory. Many OWL (RDF) stores are available, such as Jena TDB, OWLIM, AlegroGraph, KAON2, Oracle database 11g and PelletDB. Many in-memory reasoners also support database-backed reasoning by connecting to database enabled OWL frameworks. For example, FaCT++ and CEL can be plugged into OWL API for which OWLDB [48] is a de facto database backend.

9) *Ontology Manipulation*

Applications that need to modify ontologies (i.e. add/remove axioms), e.g. ontology editing tools, may benefit from a set of powerful ontology manipulation interfaces in the underlying reasoner. Many state of the art reasoners either have a native rich set of ontology manipulation interfaces, e.g. KAON2, or can be integrated with an ontology manipulation framework, e.g. OWL API and Jena, providing a well-integrated ontology manipulation/reasoning environment. Some other reasoners, such as Bossam, have relative simple (or even no) native ontology manipulation interfaces and are also not integrated into any ontology manipulation frameworks, so they are not quite suitable for this characteristic.

10) *Explanation of Deductions, Debugging*

Explanation of deductions and debugging are required by some applications such as ontology engineering tools, configuration management tools [1, 15] or bioinformatics applications. Some reasoners, such as Pellet, and Jena, implement native explanation components enabling justifications to be derived for inferences. OWL API also implements black-box debugging mechanisms [49], which therefore allows explanations to be generated for all pluggable reasoners even without built-in explanation components.

11) *Miscellaneous*

There are some other characteristics may affect the selection of an appropriate reasoner such as UI (e.g. command-line, API, GUI), remote interface (DIG, self-defined), operating platforms (J2SE, J2ME CDC, J2ME CLDC, C++, prolog), open source support, pricing, in active development, and so on. They are not discussed in detail in this paper. However, we assert that the RESP process provides native support to be extended to support additional or alternative ACs and RCs.

IV. The TARS Reasoner Selection Tool

A prototype implementation of RESP, termed TARS (Tool for Automatic Reasoner Selection), is constructed as a desktop application. The above identified candidate ACs are incorporated into TARS and can be selected on an AC selection interface. A hint is provided for users to view an explanation for each AC. Five candidate reasoners, i.e. FaCT++, KAON2, Pellet, Jena and COROR [27], are registered in TARS and stored locally so they can be re-used. Their selection is motivated by the fact that they are from different reasoner categories and therefore have different RCs. Results are displayed using traffic light notation. For each AC users can examine why each reasoner is / is not suitable. The matching process is currently hard coded in Java, however ongoing work is focussing on using a generalized rule-based approach to perform the matching.

V. Evaluation

This section discusses the design and results of the usability experiment of RESP.

A. *Overview*

Two different tasks, i.e. a reasoner selection task and a reasoner registration task, are designed to allow evaluation participants to experience both distinct facets of RESP. The reasoner selection task requires participants to select a most appropriate reasoner for a given application scenario using the RESP process with TARS. In this task participants are provided with an application scenario, from which they must identify the ACs, and then input them into TARS to select an appropriate reasoner. For the given scenario successful identification of all ACs leads to the selection of Pellet and an incomplete (but correct) AC set will limit the number of candidate reasoners. Participants can iteratively refine the selected ACs until the complete set of ACs is achieved. The aims of this experiment are (1) to investigate the difficulties for participants to identify the complete correct set of ACs, and (2) to support participants using the RESP process to select a reasoner for the given application scenario and collect feedback on its usability. A second task requires participants to analyze a description of an existing reasoner implementation authored by reasoning experts, and register its RCs using TARS.

In total 22 participants with software engineering experience took part in the experiment. They were divided into two groups according to specialities: an *application-aware group* with 17 participants experienced in ontology-based applications and a *reasoning-aware group* with 5 participants with stronger backgrounds in OWL reasoners and reasoning algorithms. The small size of the reasoning-aware group is reasonable as the reasoner registration task aims to collect expertise rather than large volumes of feedback. The reasoner selection task was assigned to the application-aware group and the reasoner registration task to the reasoning-aware group. Two questionnaire types were used to gather feedback and comments: a RESP evaluation questionnaire to gather feedback on RESP and TARS, and a SUS questionnaire as a generally accepted approach to evaluate the usability.

B. *Results and Discussions*

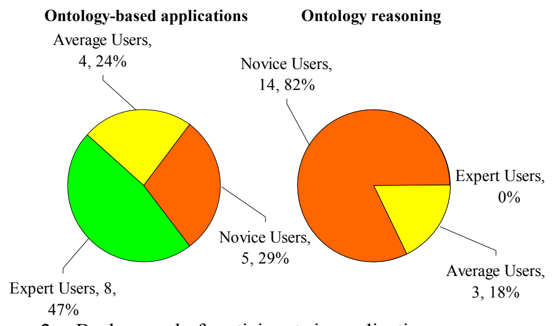

Figure 2. Background of participants in application-aware group

Self-assessment questions were used allowing participants in the application-aware group to rate their expertise in ontology-based applications and ontology reasoning (Figure 2). 12/17 participants (71%) had good knowledge of ontology-based applications (expert and average users) and nearly half of the group (8, 47%) were expert users. However most (14, 82%) participants had little or no knowledge on the specifics of ontology reasoning and only 3 participants (18%) had good reasoning-specific knowledge.

Results of the reasoner selection task indicates all participants successfully identified the most appropriate reasoner for the given application scenarios in two iterations despite most having little knowledge of the specifics of ontology reasoning. Among them, 12/17 participants limited the reasoner selection results to only 2 candidate reasoners in the first iteration, and with just a little help refined their ACs to identify the correct reasoner in the second iteration. The other 5 participants selected the correct reasoner in the first iteration.

AC entry iterations were tracked and causes for mis-selections were gathered. Investigations reveal two major findings as follows. Firstly most participants were not familiar with the DL approach to expressing ontology expressivity leading to failures identifying the correct reasoner. It is then reasonable to infer that with this issue solved, e.g. by carefully explaining DL expressivity notations, using OWL sublanguages to represent ontology expressivity, or by automatically determining the expressivity of a given application knowledge base, more participants can pinpoint the correct reasoner using only one iteration. Secondly, the AC described as "*reasoning over databases*" was erroneously selected by 10 participants due simply to presentation inconsistencies in the scenario. This problem can be easily rectified by providing some additional assistance to determine if this is necessary (e.g. based on the size of the data store). For both of these issues no change of the RESP process is required.

Results of the RESP evaluation questionnaire show positive feedback. Most participants agree that RESP is easy to understand, the given set of ACs can precisely capture most applications, and that the TARS interfaces were easy to use. 16 out of the 17 participants from the application-aware group agree that RESP is useful in helping them to choose the most appropriate gold-standard reasoner for their given application. SUS Scores from the questionnaires were calculated using the approach given in [9]. The mean score was 79/100 for the application-aware group and 81/100 for the reasoning-aware group. According to [10] such SUS scores indicate the usability of RESP is between good (71.4) and excellent (85.5).

The above discussion indicates that in spite of deficiencies and shortcomings of the TARS prototype, the RESP process still shows its usefulness in helping users with little knowledge of ontology reasoning to choose a most appropriate reasoner for applications. Results show even without the identification of a full set of ACs RESP is still able to greatly reduce the number of candidate reasoners. Therefore it can serve as a pre-selection tool of the consultation-based approach thereby reducing cost and effort.

For the reasoner registration task all participants successfully identified the complete set of RCs and registered them using TARS. However most of them thought the reasoner registration UI of TARS was too cumbersome to use, therefore requiring further work. However it is envisioned that the reasoner registration interface will not be frequently used.

Comments mainly focused on difficulties understanding and identifying some ACs, and the relatively immature prototype TARS interface. However these issues can be partly solved using more detailed explanation notes and better organized UI layout. Comments and suggestions for improvements of RESP and TARS include: using pre-defined profiles for applications to reduce the efforts required to identify ACs, using a more complex selection mechanism such as a case-based selection mechanism which takes ontologies, queries, and expected results as inputs to perform selection.

VI. Conclusion and Future Work

This paper aims to address the problem that the rapid development of both the OWL reasoning technologies and semantic applications will require more efforts to select an appropriate reasoner for semantic applications. To address this we present RESP, a computer aided process to assist application developers to select an appropriate OWL reasoner for their application. Furthermore a set of example ACs is identified and discussed to demonstrate and evaluate this process. A prototype implementation of RESP, TARS, was built enabling users to select the most appropriate reasoner by following RESP, and to register new candidate reasoners.

Although a relatively simple match-based approach was used in RESP, the usability experiment shows good usability. All participants (most of whom have little knowledge of ontology reasoning) can limit the set candidate reasoners to a small subset and some participants independently selected the gold-standard reasoner. Further analysis indicates that with a better interface and tool support more participants are likely to independently identify the appropriate reasoner. Feedback in the RESP evaluation questionnaire indicates 94% of participants agree that RESP is useful in helping them to choose an appropriate reasoner. SUS scores of 79/100 and 80/100 indicates its good usability.

This research is still at a preliminary stage and future work includes five aspects. First the identified ACs and RCs are still relatively simplistic and incomplete for real world scenarios. Secondly a more complex and extensible selection mechanism can be designed to complement the existing match-based selection. Thirdly the performance aspect of ACs needs to be considered as well. Fourthly as a possible way to extend this approach profiles for different applications can be constructed reducing the effort required for to identify ACs, and finally, making RESP web-accessible and extending it to support OWL 2 reasoners and applications.

Acknowledgment

This work is supported by the Irish Government in "Network Embedded Systems" (NEMBES), under the Higher Education Authority's Programme for Research in Third Level Institutions (PRTLI) cycle 4, and by Science Foundation Ireland in "Federated, Autonomic Management of End-to-End Communications Services" (FAME) (grant 08/SRC/I1403).

References

[1] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, The Description Logic Handbook: Theory, Implementation and Applications, 2nd ed.: Cambridge University Press New York, 2007.

[2] M. Calder, R. A. Morris, and F. Peri, "Machine reasoning about anomalous sensor data," Ecological Informatics, vol. 5, 2010, pp. 9-18.

[3] C. Halaschek-Wiener and V. Kolovski, "Syndication on the Web using a Description Logic Approach," in J. Web Semantics Science Services and Agents on the World Wide Web, vol. 6, issue 3, pp. 171-190, 2008.

[4] M. Gomez, A. Preece, M. P. Johnson, G. d. Mel, W. Vasconcelos, C. Gibson, A. Bar-Noy, K. Borowiecki, T. L. Porta, D. Pizzocaro, H. Rowaihy, G. Pearson, and T. Pham, "An Ontology-Centric Approach to Sensor-Mission Assignment," in Proc. Intl. Conf. on Knowledge Engineering and Knowledge Management, 2008.

[5] M. A. Harris, J. Clark, A. Ireland, J. Lomax, M. Ashburner, R. Foulger, K. Eilbeck, S. Lewis, B. Marshall, and C. Mungall, "The Gene Ontology

(GO) database and informatics resource," Nucleic acids research, vol. 32, pp. D258 - D261 (2004)

[6] V. Haarslev and R. Möller, "RACER system description," In Proc. Intl. Joint Conf. on Automated Reasoning, pp. 701-706, 2001.

[7] V. Haarslev and R. Möller, "Incremental Query Answering for Implementing Document Retrieval Services," in Proc. Intl. Workshop on Description Logics, pp. 85-94, 2003.

[8] C. Halashek-Wiener, B. Parsia, and E. Sirin, "Description Logic Reasoning with Syntactic Updates," in Proc. Intl. Conf. on Ontologies, Databases, and App. of Semantics, pp. 722-737, 2006.

[9] J. Brooke, "SUS-A quick and dirty usability scale," Usability evaluation in industry, pp. 189-194, 1996.

[10] A. Bangor, P. Kortum, and J. Miller, "Determining What Individual SUS Scores Mean: Adding an Adjective Rating Scale," J. of Usability Studies, vol. 4, pp. 114-123, 2009.

[11] C. M. Keet, M. Roos, and M. S. Marshall, "A Survey of Requirements for Automated Reasoning Services for Bio-Ontologies in OWL," in Proc. Intl. Workshop on OWL: Experiences and Directions, 2007.

[12] B. Motik and U. Sattler, "A comparison of reasoning techniques for querying large description logic aboxes," in Proc. Intl. Conf. on Logic for Programming, Artificial Intelligence, and Reasoning, 2006.

[13] B. Parsia, C. Halaschek-wiener, and E. Sirin, "E.S.: Towards Incremental Reasoning Through Updates," in OWL DL, in Proc Intl. Conf. on World Wide Web, 2006.

[14] D. J. Russomanno, C. R. Kothari, and O. A. Thomas, "Building a Sensor Ontology: A Practical Approach Leveraging ISO and OGC Models," in Proc. of Intl. Conf. on Artificial Intelligence, pp637-643, 2005.

[15] H. H. Shahri, J. A. Hendler, and A. A. Porter, "Software configuration management using ontologies," in Proc. Intl. Workshop on Semantic Web Enabled Software Engineering, 2007.

[16] A. Sheth, C. Henson, and S. S. Sahoo, "Semantic Sensor Web," in IEEE Internet Computing Magazine. vol. 12, pp. 78-83, 2008.

[17] D. Tsarkov, A. Riazanov, S. Bechhofer, and I. Horrocks, "Using Vampire to Reason with OWL," in Proc Intl. Semantic Web Conf., 2004.

[18] M. Ushchold, P. Clark, F. Dickey, C. Fung, S. Smith, S. Uczekaj, M. Wilke, S. Bechhofer, and I. Horrocks, "A semantic infosphere," in Proc. Intl. Semantic Web Conf., pp. 882-896, 2003.

[19] Y. Zou, T. Finin, and H. Chen, "F-OWL: an Inference Engine for Semantic Web," in Proc. Workshop on Formal Approaches to Agent-Based Systems (FAABS III), pp. 16-18, 2004.

[20] M. Compton, C. Henson, L. Lefort, H. Neuhaus, and A. Sheth, "A Survey of the Semantic Specification of Sensors," in Proc. Intl. Workshop on Semantic Sensor Networks, 2009.

[21] J. Keeney, D. Roblek, D. Jones, D. Lewis, D. O'Sullivan, "Extending Siena to support more expressive and flexible subscriptions", in Proc. Intl. Conf. on Distributed Event-Based Systems, 2008.

[22] C. Forgy, "Rete: A Fast Algorithm for the many pattern/many object pattern match problem," Artificial Intelligence, vol. 19, pp. 17-37, 1982.

[23] B. C. Grau, C. Halaschek-Wiener, Y. Kazakov and B. Suntisrivaraporn, "Incremental Classification of Description Logics Ontologies, " J. of Automated Reasoning, vol. 44, pp. 337-369, 2010.

[24] H. Stuckenschmidt, S. Ceri, E. D. Valle and F. v. Harmelen, "Towards Expressive Stream Reasoning," in Proc. the Dagstuhl Seminar on Semantic Aspects of Sensor Networks, 2010.

[25] D. F. Barbieri, D. Braga, S. Ceri, E. D. Valle, and M. Grossniklaus, "Incremental Reasoning on Streams and Rich Background Knowledge," in Proc. Extended Semantic Web Conference, pp. 1-15, 2010.

[26] S. Bechhofer, I. Horrocks, and D. Turi, "The OWL Instance Store: System Description," in Proc. of Intl. Conf. Automated Deduction, 2005.

[27] W. Tai, J. Keeney, and D. O'Sullivan, "COROR: A COmposable Rule-entailment Owl Reasoner for Resource Constrained Devices," in Proc. 5th Intl. Symposium on Rules: Research Based and Industry Focused, 2011.

[28] G. Meditskos and N. Bassiliades, "DLEJena: A Practical Forward-Chaining OWL 2 RL Reasoner Combining Jena and Pellet," J. Web Sematnics: Science, Services and Agends on the WWW, vol. 8, 2010.

[29] B. Bishop, A. Kiryakov, D. Ognyanoff, I. Peikov, Z. Tashev, and R. Velkov, "OWLIM: A Family of Scalable Semantic Repositories," Semantic Web Journal, To be appear, 2011.

[30] H. J. ter Horst, "Completeness, decidability and complexity of entailment for RDF Schema and a semantic extension involving the OWL vocabulary," J. Web Semantics: Science, Services and Agents on the WWW, vol. 3, pp. 79-115, 2005.

[31] F. Baader, S. Brandt, and C. Lutz, "Pushing the EL Envelop Further," in Proc. Intl. Workshop on OWL: Experiences and Directions, 2008.

[32] M. Stocker and M. Smith, "Owlgres: A Scalable OWL Reasoner,"in Proc. Intl. Workshop on OWL:Experiences and Directions, 2008.

[33] A. Acciarri, D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, M. Palmieri, and R. Rosati, "QuOnto: Querying Ontologies," in Proc. National Conf. on Artificial Intelligence, 2005.

[34] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati, "Tractable Reasoning and Efficient Query Answering in Description Logics: The DL-Lite Family," J. Automated Reasoning, vol. 39, 2007.

[35] M. Eid, R. Liscano, and A. El Saddik, "A Universal Ontology for Sensor Networks Data," in Proc. Intl. Conf. on Computational Intelligence for Measurement Systems and Applications, 2007.

[36] J.-H. Kim, H. Kwon, D. -H. Kim, H. –Y. Kwak, and S. –J. Lee, "Building a Service-Oriented Ontology for Wireless Sensor Networks," in Proc. Intl. Conf. on Computer and Information Science, 2008.

[37] J. Mendez and B. Suntisrivaraporn, "Reintroducing CEL as an OWL 2 EL Reasoner", in Proc. Intl. Workshop on Description Logic, 2009.

[38] V. Vassiliadis, J. Wielemaker, and C. Mungall, "Processing OWL2 ontologies using Thea: An application of logic programming," in Proc. Intl. Workshop on OWL: Experiences and Directions, 2009.

[39] D. F. Barbieri, D. Braga, S. Ceri and M. Grossniklaus, "An execution environment for C-SPARQL queries," in Proc. Intl. Conf. on Extending Database Technology, 2010.

[40] M. Compton, H. Neuhaus, K. Taylor, K. -N. Tran, "Reasoning about Sensors and Compositions," in Proc. Intl. Workshop on Semantic Sensor Networks, 2009.

[41] R. Brennan, W. Tai, D. O'Sullivan, M. S. Aslam, S. Rea, and D. Pesch, "Open Framework Middleware for Intelligent WSN Topology Adaption in Smart Buildings," in Proc. Intl. Conf. on Ultra Modern Telecommunications & Workshops, 2009.

[42] V. Haarslev, and R. Möller, "Practical Reasoning in RACER with a Concrete Domain for Linear Inequations", in Proc. Intl. Workshop on Description Logics, 2002.

[43] C. Lutz, "Reasoning with Concrete Domains," in Proc. Intl. Joint Conf. on Artificial Intelligence, 1999.

[44] U. Hustadt, B. Motik and U. Sattler, "Reasoning in Description Logics with a Concrete Domain in the Framework of Resolution," in Proc. European Conf. on Artificial Intelligence, 2004.

[45] V. Haarslev, and R. Möller, "Description Logic Systems with Concrete Domains: Applications for the Semantic Web," in Proc. Intl. Workshop on Knowledge Representation meets Databases, 2003.

[46] Y. Katz, and B. Parsia, "Towards a Nonmonotonic Extension to OWL," in Proc. Intl. Workshop on OWL: Experiences and Directions, 2005.

[47] J. Tao, E. Sirin, J. Bao, D. L. McGuinness, "Integrity Constraints in OWL," in Proc. of AAAI Conf. on Artificial Intelligence, 2010.

[48] J. Henss, J. Kleb, S. Grimm, and J. Bock, "A Database Backend for OWL," in Proc. Intl. Workshop on OWL: Experiences and Directions, 2009.

[49] A. Kalyanpur, B. Parsia, and E. Sirin, "Debugging Unsatisfiable Classes in OWL Ontologies," J. Web Semantics, vol.3, issue 4, 2006.

[50] B. Motik, R. Shearer, and I. Horrocks, "Hypertableau Reasoning for Description Logics," J. Artificial Intelligence Research, vol. 36, 2009.

[51] M. Krötzsch, A. Mehdi, and S. Rudolph, "Orel: Database-driven reasoning for OWL 2 profile," in Proc. Intl Workshop on Description Logic, 2010.

[52] D. Cleary and B. Danev, "Using ontologies to simplify wireless network configuration," in Proc. of the FOMI Workshop, 2005.